$1.20

Apple Assembly Line

Volume 2 -- Issue 3 December, 1981

In This Issue...

EXCEL-9: A 6809 Card with FLEX.............Bob Sander-Cederlof

For the last month and a half I have been working with a fantastic new device: the EXCEL-9 from Seikou Electronics in Japan. The EXCEL-9 contains a 6809E CPU, 8K bytes of ROM, and an interval timer. The 8K ROM contains a monitor with 35 commands (including mini-assembler anddis-assembler commands). The introductory price of $399.95 includes the FLEX Operating System from Technical Systems Consultants (TSC), with utilities, text editor, and macro assembler.

The board will soon be appearing in your local computer stores, courtesy of ESD Laboratories. I worked with them to translate the excellent reference manual into English. (That explains how I obtained one of the boards so early.)

EXCEL-9 has a lot of unique features that should make it a very popular board:

* An on-board interval timer (with 24 intervals from 2 microseconds to 16 seconds) can be used from both the 6809 and 6502.

* Built-in linkage routines for calling 6809 subroutines from Applesoft, Integer BASIC, or 6502 machine language. You can also call 6502 routines and even DOS 3.3 commands from 6809 programs.

* Option of using standard Apple intelligent interfaces with 6502 firmware, or of using new cards with 6809 firmware.

* Memory Mapping that supports the FLEX operating system. Future option to add external memory to EXCEL-9, allowing full-speed multiprocessing.

I intend to handle these boards. You can order them from me now, but please allow a while for delivery. The documentation is ready for the printer, but not yet printed.

Applesoft Hi-Res Subroutines................Bob Sander-Cederlof

One of the questions I hear the most is "How can I call the Hi-Res subroutines in the Applesoft ROMs?" The basic information about those subroutines has been published (in Apple Orchard, Vol. 1 No. 1), but with an error in the subroutine addresses.

First, some important locations in page zero:

$1A,1B	Shape pointer used by DRAW and XDRAW
$1C	Last used color byte
$26,27	Address of byte containing X,Y point
$30	Bit mask for bit in that byte
$E0,E1	X-coordinate (0-279)
$E2	Y-coordinate (0-191)
$E4	Color
$E6	Page ($20 if HGR, $40 if HGR2)
$E7	SCALE= value
$E8,E9	Address of beginning of shape table
$EA	Collision counter
$F9	ROT= value

The software uses some other page zero variables, but I am not too clear yet on their purpose.

Now here are the major entry points:

HGR2	$F3D8	Initialize and clear hi-res page 2.
HGR	$F3E2	Initialize and clear hi-res page 1.
HCLR	$F3F2	Clear the current hi-res screen to black.
BKGND	$F3F6	Clear the current hi-res screen to the last plotted color (from ($1C).
HPOSN	$F411	Positions the hi-res cursor without plotting a point. Enter with (A) = Y-coordinate, and (Y,X) = X-coordinate.
HPLOT	$F457	Calls HPOSN and tries to plot a dot at the cursor's position. If you are trying to plot a non-white color at a complementary color position, no dot will be plotted.
HLIN	$F53A	Draws a line from the last plotted point or line destination to: (X,A) = X-coordinate, and (Y) = Y-coordinate.
HFIND	$F5CB	Converts the hi-res coursor's position back to X- and Y-coordinates; stores X-coordinate at $E0,E1 and Y-coordinate at $E2.

DRAW	$F601	Draws a shape. Enter with (Y,X) = the address of the shape table, and (A) = the rotation factor. Uses the current color.
XDRAW	$F65D	Draws a shape by inverting the existing color of the dots the shape draws over. Same entry parameters as DRAW.
SETHCOL	$F6EC	Set the hi-res color to (X), where (X) must be between 0 and 7.

I wrote a sample demonstration program of the hi-res subroutines. First, here is an Applesoft version. Note that it first sets the whole screen to a particular color, and then draws a series of nested squares in a complementary color. Since it is nice and short, why don't you type it in and try it?

```
100  HGR2
110  FOR C = 0 TO 7: HCOLOR= C
120  HPLOT 0,0: CALL 62454: REM CLEAR TO CURRENT COLOR
130  HCOLOR= 7 - C
140  FOR S = 10 TO 190 STEP 10
150 X1 = 140 - S / 2:X2 = X1 + S
160 Y1 = 95 - S / 2:Y2 = Y1 + S
170  HPLOT X1,Y1 TO X2,Y1 TO X2,Y2 TO X1,Y2 TO X1,Y1
180  NEXT S
190  PRINT  CHR$ (7): FOR I = 1 TO 500: NEXT
200  NEXT C
210  TEXT
```

Now here is the assembly language program for the same task. It seemed to run about twice as fast as the Applesoft version, but I didn't use the stopwatch on it.

```
                  1000 *-------------------------------
                  1010 *      SAMPLE PLOTTING PROGRAM
                  1020 *-------------------------------
001C-             1030 AS.LASTCLR .EQ $1C
                  1040 *-------------------------------
F3D8-             1050 AS.HGR2    .EQ $F3D8 SET UP HI-RES PAGE 2
F3F2-             1060 AS.HCLR    .EQ $F3F2 CLEAR HI-RES SCREEN
F3F6-             1070 AS.BKGND   .EQ $F3F6 CLEAR HI-RES SCREEN TO LAST COLOR
F411-             1080 AS.HPOSN   .EQ $F411 MOVE CURSOR TO (Y,X),(A)
F457-             1090 AS.HPLOT   .EQ $F457 PLOT A DOT AT (Y,X),(A)
F53A-             1100 AS.HLIN    .EQ $F53A DRAW A LINE FROM LAST POINT TO (X,A),(Y)
F6EC-             1110 AS.SETHCOL .EQ $F6EC SET HI-RES COLOR
FB2F-             1120 MON.TEXT   .EQ $FB2F
                  1130 *-------------------------------
                  1140 HI.RES.DEMO
0800- 20 D8 F3    1150        JSR AS.HGR2
0803- A2 00       1160        LDX #0       FOR COLOR = 0 TO 7
0805- 8E AD 08    1170 .1     STX COLOR
0808- 20 EC F6    1180        JSR AS.SETHCOL
080B- 85 1C       1190        STA AS.LASTCLR
080D- 20 F6 F3    1200        JSR AS.BKGND   CLEAR SCREEN TO SOLID COLOR
0810- AD AD 08    1210        LDA COLOR
0813- 49 07       1220        EOR #7       COMPLEMENTARY COLOR
```

```
0815- AA       1230        TAX
0816- 20 EC F6 1240        JSR AS.SETHCOL
0819- 20 28 08 1250        JSR DRAW.SQUARE
081C- AE AD 08 1260        LDX COLOR      NEXT COLOR
081F- E8       1270        INX
0820- E0 08    1280        CPX #8
0822- 90 E1    1290        BCC .1
0824- 20 2F FB 1300        JSR MON.TEXT
0827- 60       1310        RTS
               1320 *--------------------------------
               1330 DRAW.SQUARE
0828- A9 0A    1340        LDA #10      FOR SIZE=10 TO 190 STEP 10
082A- 8D AE 08 1350 .1     STA SIZE
082D- 4A       1360        LSR          SIZE/2
082E- 8D AF 08 1370        STA SIZE2
0831- A9 00    1380        LDA #0
0833- 8D B1 08 1390        STA XSTART+1
0836- 8D B4 08 1400        STA XSTOP+1
0839- 38       1410        SEC          XSTART=140-SIZE/2
083A- A9 8C    1420        LDA #140
083C- ED AF 08 1430        SBC SIZE2
083F- 8D B0 08 1440        STA XSTART
0842- 18       1450        CLC          XSTOP=XSTART+SIZE
0843- 6D AE 08 1460        ADC SIZE
0846- 8D B3 08 1470        STA XSTOP
0849- 38       1480        SEC
084A- A9 5F    1490        LDA #95      YSTART=95-SIZE/2
084C- ED AF 08 1500        SBC SIZE2
084F- 8D B2 08 1510        STA YSTART
0852- 18       1520        CLC          YSTOP=YSTART+SIZE
0853- 6D AE 08 1530        ADC SIZE
0856- 8D B5 08 1540        STA YSTOP
0859- AC B1 08 1550        LDY XSTART+1 HPLOT XSTART,YSTART
085C- AE B0 08 1560        LDX XSTART
085F- AD B2 08 1570        LDA YSTART
0862- 20 57 F4 1580        JSR AS.HPLOT
0865- AE B4 08 1590        LDX XSTOP+1    TO XSTOP,YSTART
0868- AD B3 08 1600        LDA XSTOP
086B- AC B2 08 1610        LDY YSTART
086E- 20 3A F5 1620        JSR AS.HLIN
0871- AE B4 08 1630        LDX XSTOP+1    TO XSTOP,YSTOP
0874- AD B3 08 1640        LDA XSTOP
0877- AC B5 08 1650        LDY YSTOP
087A- 20 3A F5 1660        JSR AS.HLIN
087D- AE B1 08 1670        LDX XSTART+1   TO XSTART,YSTOP
0880- AD B0 08 1680        LDA XSTART
0883- AC B5 08 1690        LDY YSTOP
0886- 20 3A F5 1700        JSR AS.HLIN
0889- AE B1 08 1710        LDX XSTART+1   TO XSTART,YSTART
088C- AD B0 08 1720        LDA XSTART
088F- AC B2 08 1730        LDY YSTART
0892- 20 3A F5 1740        JSR AS.HLIN
0895- 18       1750        CLC
0896- AD AE 08 1760        LDA SIZE     NEXT SIZE
0899- 69 0A    1770        ADC #10
089B- C9 BF    1780        CMP #191
089D- 90 8B    1790        BCC .1
               1800 DELAY.LOOP
089F- A0 00    1810        LDY #0       DELAY LOOP SO WE CAN SEE IT
08A1- A2 00    1820 .1     LDX #0
08A3- CA       1830 .2     DEX
08A4- D0 FD    1840        BNE .2
08A6- AD 30 C0 1850        LDA $C030    AND HEAR IT
08A9- 88       1860        DEY
08AA- D0 F5    1870        BNE .1
08AC- 60       1880        RTS
               1890 *--------------------------------
08AD-          1900 COLOR  .BS 1
08AE-          1910 SIZE   .BS 1
08AF-          1920 SIZE2  .BS 1
08B0-          1930 XSTART .BS 2
08B2-          1940 YSTART .BS 1
08B3-          1950 XSTOP  .BS 2
08B5-          1960 YSTOP  .BS 1
```

S-C ASSEMBLER II Version 4.0.................................$55.00

Includes Manual, Diskette with Assembler and sample source programs, and Quick Reference Card.

Source code of Version 4.0 on disk..............................$95.00

Fully commented, easy to understand and modify to your own tastes.

Cross Assembler Patches for 6809................................$20.00

Requires possession of Version 4.0. Enables you to develop programs for the Motorola 6809 CPU. (The MILL from Stellation, EXCEL-9 from ESD Laboratories, or the Radio Shack Color Computer.)

Cross Assembler for 6800..$22.50

Requires possession of Version 4.0. Enables you to develop programs for the Motorola 6800, 6801, and 6802 CPUs.

AAL Quarterly Disks.................................each $15.00

Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1:	Oct - Dec 1980	QD#4:	Jul - Sep 1981
QD#2:	Jan - Mar 1981	QD#5:	Oct - Dec 1981
QD#3:	Apr - Jun 1981		

Double Precision Floating Point for Applesoft...................$50.00

Provides 21-digit precision for Applesoft programs. Includes subroutines for standard math functions.

Some Simple Integer BASIC Games.................................$10.00

Includes 4x4x4 tic-tac-toe, lo-res space war, lo-res jig-saw puzzle, and mastermind.

Blank Diskettes.............................package of 20 for $50.00

Verbatim Datalife, with hub rings, no labels, in plain white jackets, in cellophane wrapper.

Lower-Case Display Encoder ROM..................................$25.00

Works only Revision level 7 Apples. Replaces the encoder ROM. Comes with instructions.

Diskette Mailing Protectors.....................10-99: 40 cents each
100 or more: 25 cents each

Corrugated folder specially designed for mailing mini-floppy diskettes. Fits in standard 6x9-inch envelope. (Envelopes 5-cents each, if you need them.)

Zip-Lock Bags (2-mil, 6"x9")............................100 for $8.50
(2-mil, 9"x12")..........................100 for $13.00

Books, Books, Books......................compare our discount prices!

"Beneath Apple DOS", Worth & Lechner.............	($19.95)	$18.00
"What's Where in the Apple", William Leubert.....	($14.95)	$14.00
"6502 Assembly Language Programming", Leventhal..	($16.99)	$16.00
"Apple Assembly Language", Don & Kurt Inman......	($12.95)	$12.00

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050   We take Master Charge and VISA ***

HEX CONSTANTS IN APPLESOFT

David H Bartley

Coding in BASIC has several frustrations for the assembly language programmer. One small but constant irritant for me has been the inability to directly specify hexadecimal values in Applesoft statements or in response to an INPUT command. I finally decided to do something about it when I read Bob Sander-Cederlof's article on the CHRGET routine in the September **Apple Assembly Line**. The result is the short program shown in Listing 1.

My goal was to be able to enter a hex constant, defined as a "$" followed by one or more hex digits, anywhere Applesoft would allow an integer constant to appear. I nearly succeeded -- I'll discuss the exceptions a little later. I now can write statements like

```
100 FOR I = $0 TO $FF
110 INPUT X, Y
120 Z(I) = $100*X + Y - $3DEF
```

The responses to the INPUT statement may also be hex constants. Values may range from -$FFFF (-65535) to $FFFF (+65535); the left-most bit is <u>not</u> considered a sign bit.

My program is set up by BRUN-ning the object file XB.A/S HEX CONSTANTS (see line 1010). Initialization consists of modifying the Applesoft CHRGET routine to branch into new code starting at line 1400. As you may recall, CHRGET is used by the BASIC interpreter to fetch characters and tokens from the program text or keyboard when a program is executing. The new CHRGET code watches for a "$" character; when one is found, it scans forward until it hits a character which is not a hex digit, converting to a binary value (in VAL) on the fly.

Variable IDX serves two purposes. It is normally negative, signifying that characters are to be fetched without special action until a "$" is encountered. After a hex constant is found and converted to a binary value, IDX becomes a positive index into a power-of-ten table to facilitate converting VAL to a decimal value. Each subsequent call to CHRGET then returns a successive character of the decimal integer representation of VAL until IDX becomes -1, the entire value has been transformed from hex to decimal, and the normal mode is restored.

There are, of course, several complications. One is the BASIC "DEF" command, which happens to consist of a string of hex digits. Applesoft therefore parses a constant like "$3DEF" as the ASCII characters "$" and "3" followed by the DEF token (hex 88). Lines 1760 to 1840 take care of that.

A more serious complication is the existence of a frequently used alternate entry point to CHRGET called CHRGOT. CHRGOT is called to fetch the previous item from the text rather than the next one. It seems that numeric constants are parsed from several places within the Applesoft interpreter, with some using CHRGOT and others not. When I fixed things up so CHRGOT would work for inline constants and the INPUT command, it no longer worked for values in DATA statements (or for hex line numbers, for that matter!)

The trick that makes CHRGOT work (most of the time) is to back up TXTPTR and then return a leading zero to start off the converted decimal value. The zero causes no consternation for the parts of the interpreter that see it and is not missed by those that don't. If CHRGOT is not called, however, TXTPTR should not be backed up. You can't win!

I hope others will be able to make use of this routine -- better, that someone will overcome the problem with DATA statement values. It has been quite valuable to me as it is, as well as quite an education in understanding the inner workings of the Applesoft interpreter.

XS.A/S HEX CONSTANTS   8 November 1981   David H Bartley

```
                  1ØØØ   .OR $Ø3ØØ
                  1Ø1Ø   .TF XB.A/S HEX CONSTANTS
                  1Ø2Ø *-------------------------------
                  1Ø3Ø *
                  1Ø4Ø *    APPLESOFT HEX CONSTANTS
                  1Ø5Ø *
                  1Ø6Ø *  WRITTEN BY DAVID H BARTLEY
                  1Ø7Ø *  AUSTIN, TEXAS -- AUGUST 1981
                  1Ø8Ø *
                  1Ø9Ø *  TO INITIALIZE:
                  11ØØ *      BRUN THIS PROGRAM
                  111Ø *
                  112Ø *  TO USE:
                  113Ø *      PRECEDE HEX CONSTANTS
                  114Ø *      WITH A "$" CHARACTER
                  115Ø *
                  116Ø *-------------------------------
EØØ3-             117Ø BASIC  .EQ $EØØ3
ØØB1-             118Ø CHRGET .EQ $ØØB1     A/S CHRGET RTN
ØØB7-             119Ø CHRGOT .EQ $ØØB7     A/S CHRGOT RTN
ØØBA-             12ØØ CHRCHK .EQ CHRGOT+3
ØØB8-             121Ø TXTPTR .EQ $B8       A/S TEXT PTR
E8D5-             122Ø OVERR  .EQ $E8D5     OVERFLOW ERROR
ØØFC-             123Ø TEMP   .EQ $FC       16 BIT TEMPORARY
ØØFE-             124Ø VAL    .EQ $FE       16 BIT VALUE
                  125Ø *-------------------------------
                  126Ø INIT
Ø3ØØ- A9 4C       127Ø        LDA #$4C      MODIFY CHRGET
Ø3Ø2- 85 B1       128Ø        STA CHRGET    TO CALL HERE
Ø3Ø4- A9 18       129Ø        LDA #NEW.CHRGET
Ø3Ø6- 85 B2       13ØØ        STA CHRGET+1
Ø3Ø8- A9 Ø3       131Ø        LDA /NEW.CHRGET
Ø3ØA- 85 B3       132Ø        STA CHRGET+2
Ø3ØC- 4C Ø3 EØ    133Ø        JMP BASIC     RETURN TO A/S
                  134Ø NEXTCH
Ø3ØF- E6 B8       135Ø        INC TXTPTR    DUPLICATE THE
Ø311- DØ Ø2       136Ø        BNE .1Ø       OLD CHRGET
Ø313- E6 B9       137Ø        INC TXTPTR+1
Ø315- 4C B7 ØØ    138Ø .1Ø    JMP CHRGOT
                  139Ø *-------------------------------
                  14ØØ NEW.CHRGET
Ø318- 2C C9 Ø3    141Ø        BIT IDX       NORMAL MODE?
Ø31B- 1Ø 5C       142Ø        BPL .6Ø       -NO
                  143Ø *
                  144Ø * CHECK FOR "$" AS NEXT CHARACTER
                  145Ø *
Ø31D- 2Ø ØF Ø3    146Ø        JSR NEXTCH    GET CHAR
Ø32Ø- C9 24       147Ø        CMP #$24      "$" ?
Ø322- DØ 52       148Ø        BNE .5Ø       -NO, RETURN IT
                  149Ø .1Ø
                  15ØØ * PARSE A HEX NUMBER AND CONVERT
                  151Ø * IT TO A BINARY VALUE
                  152Ø *
Ø324- A9 ØØ       153Ø        LDA #Ø
```

```
Ø326- 85 FE     154Ø          STA VAL       VAL := Ø
Ø328- 85 FF     155Ø          STA VAL+1
Ø32A- A9 Ø4     156Ø          LDA #4        INDEX TO POWER
Ø32C- 8D C9 Ø3  157Ø          STA IDX       OF TEN TABLE
                158Ø .2Ø
Ø32F- 2Ø ØF Ø3  159Ø          JSR NEXTCH    GET HEX DIGIT
Ø332- FØ 3Ø     16ØØ          BEQ .4Ø       -EOL OR ":"
Ø334- 38        161Ø          SEC
Ø335- E9 3Ø     162Ø          SBC #$3Ø      CHECK FOR DIGIT
Ø337- 3Ø 18     163Ø          BMI .35       -NOT A DIGIT
Ø339- C9 ØA     164Ø          CMP #1Ø
Ø33B- 9Ø ØA     165Ø          BCC .3Ø       -OK (Ø-9)
Ø33D- E9 11     166Ø          SBC #17
Ø33F- 3Ø 23     167Ø          BMI .4Ø       -NOT A DIGIT
Ø341- C9 Ø6     168Ø          CMP #6
Ø343- BØ 1F     169Ø          BCS .4Ø       -NOT A DIGIT
Ø345- 69 ØA     17ØØ          ADC #1Ø
Ø347- 2Ø AF Ø3  171Ø .3Ø      JSR ASL4      MULT VAL BY 16
Ø34A- Ø5 FE     172Ø          ORA VAL       ADD NEW DIGIT
Ø34C- 85 FE     173Ø          STA VAL
Ø34E- 4C 2F Ø3  174Ø          JMP .2Ø
                175Ø .35
Ø351- C9 88     176Ø          CMP #$88      "DEF" TOKEN?
Ø353- DØ ØF     177Ø          BNE .4Ø       -NO
Ø355- 2Ø AF Ø3  178Ø          JSR ASL4      -YES
Ø358- A5 FE     179Ø          LDA VAL
Ø35A- Ø9 ØD     18ØØ          ORA #$ØD      ASL BY 12 AND
Ø35C- 85 FF     181Ø          STA VAL+1     ADD $ØDEF
Ø35E- A9 EF     182Ø          LDA #$EF
Ø36Ø- 85 FE     183Ø          STA VAL
Ø362- DØ CB     184Ø          BNE .2Ø       (ALWAYS)
                185Ø .4Ø
Ø364- A5 B8     186Ø          LDA TXTPTR    BACK UP THE
Ø366- DØ Ø2     187Ø          BNE .41       TEXT POINTER
Ø368- C6 B9     188Ø          DEC TXTPTR+1
Ø36A- C6 B8     189Ø .41      DEC TXTPTR
Ø36C- A5 B8     19ØØ          LDA TXTPTR    SAVE TXTPTR
Ø36E- 85 FC     191Ø          STA TEMP      IN CASE IT IS
Ø37Ø- A5 B9     192Ø          LDA TXTPTR+1  DECREMENTED
Ø372- 85 FD     193Ø          STA TEMP+1    BY THE CALLER
                194Ø *
Ø374- A9 3Ø     195Ø          LDA #$3Ø      ASCII "Ø"
Ø376- 4C BA ØØ  196Ø .5Ø      JMP CHRCHK    -EXIT
                197Ø .6Ø
                198Ø * CONVERT BINARY VALUE TO DECIMAL
                199Ø * AND RETURN THE NEXT ASCII DIGIT
                2ØØØ *
Ø379- A5 FC     2Ø1Ø          LDA TEMP      FIX ANY ATTEMPT
Ø37B- 85 B8     2Ø2Ø          STA TXTPTR    TO DECREMENT
Ø37D- A5 FD     2Ø3Ø          LDA TEMP+1    TXTPTR
Ø37F- 85 B9     2Ø4Ø          STA TXTPTR+1
Ø381- 8E CA Ø3  2Ø5Ø          STX SAVE.X
Ø384- AE C9 Ø3  2Ø6Ø          LDX IDX       POWER OF TEN
Ø387- CE C9 Ø3  2Ø7Ø          DEC IDX
Ø38A- A9 3Ø     2Ø8Ø          LDA #$3Ø      ASCII "Ø"
                2Ø9Ø .7Ø
```

```
Ø38C- 48        21ØØ          PHA            ASCII DIGIT
Ø38D- A5 FE     211Ø          LDA VAL
Ø38F- DD BF Ø3  212Ø          CMP LO.TENS,X  SET CARRY
Ø392- A5 FF     213Ø          LDA VAL+1
Ø394- FD C4 Ø3  214Ø          SBC HI.TENS,X
Ø397- 9Ø ØF     215Ø          BCC .8Ø        -EXIT LOOP
Ø399- 85 FF     216Ø          STA VAL+1
Ø39B- A5 FE     217Ø          LDA VAL
Ø39D- FD BF Ø3  218Ø          SBC LO.TENS,X
Ø3AØ- 85 FE     219Ø          STA VAL
Ø3A2- 68        22ØØ          PLA            ASCII DIGIT
Ø3A3- 18        221Ø          CLC
Ø3A4- 69 Ø1     222Ø          ADC #1         INCREMENT IT
Ø3A6- DØ E4     223Ø          BNE .7Ø        -LOOP
                224Ø .8Ø
Ø3A8- 68        225Ø          PLA            ASCII DIGIT
Ø3A9- AE CA Ø3  226Ø          LDX SAVE.X
                227Ø .9Ø
Ø3AC- 4C BA ØØ  228Ø          JMP CHRCHK     PROCESS IT
                229Ø *---------------------------------
Ø3AF- 2Ø B2 Ø3  23ØØ ASL4     JSR ASL2       ASL VAL BY 4
Ø3B2- 2Ø B5 Ø3  231Ø ASL2     JSR ASL1       ASL VAL BY 2
Ø3B5- Ø6 FE     232Ø ASL1     ASL VAL        ASL VAL BY 1
Ø3B7- 26 FF     233Ø          ROL VAL+1
Ø3B9- BØ Ø1     234Ø          BCS OVFLOW     -OVERFLOW ERROR
Ø3BB- 6Ø        235Ø          RTS            -EXIT
                236Ø OVFLOW
Ø3BC- 4C D5 E8  237Ø          JMP OVERR      REPORT OVERFLOW
                238Ø *---------------------------------
Ø3BF- Ø1        239Ø LO.TENS .DA #1
Ø3CØ- ØA        24ØØ          .DA #1Ø
Ø3C1- 64        241Ø          .DA #1ØØ
Ø3C2- E8        242Ø          .DA #1ØØØ
Ø3C3- 1Ø        243Ø          .DA #1ØØØØ
Ø3C4- ØØ        244Ø HI.TENS .DA /1
Ø3C5- ØØ        245Ø          .DA /1Ø
Ø3C6- ØØ        246Ø          .DA /1ØØ
Ø3C7- Ø3        247Ø          .DA /1ØØØ
Ø3C8- 27        248Ø          .DA /1ØØØØ
Ø3C9- FF        249Ø IDX      .DA #$FF       TABLE INDEX
Ø3CA- ØØ        25ØØ SAVE.X   .DA #Ø         SAVE X-REG
                251Ø *---------------------------------
                252Ø ZZZZZZ   .EN
```

Applesoft Line Editing Aid..................Sandy Mossberg

[Sandy is an M.D. in Port Chester, New York. You have probably seen his excellent articles and programs in NIBBLE.]

The following program is a developmental tool for line-editing Applesoft programs. It places the line you specify at the top of the screen, ready to be cursor edited. The line is displayed without added blanks at the end of each screen line, which can mess up editing of PRINT statements. Obviously, adding Konzen-like PLE features would make it much nicer, but that's a story for another day.

The program loads at the ever-popular $300. If you BRUN it, or BLOAD and CALL768, it installs itself. To use it, type a slash and a line number. For example, to edit line 150, type "/150" and a carriage return. The screen will be cleared and line 150 displayed on the top. The cursor will be placed over the first character, and you will be ready to edit it with standard cursor-editing techniques. (If there is no line 150 in memory, the bell will ring instead.)

Several aspects of the code should be of interest to assembly language programmers:

(1) As noted in AAL of 9/81, the CHRGET/CHRGOT routine screens for the command character (a slash). This technique permits concurrent use of an amper-utility. The KSW hook could be employed as yet another filter, making a trio of vectors operative.

(2) To allow "illegal" line numbers (64000-65535) to be accessed, the LINGET routine is replaced by calls to FRMEVL and GETADR (see Lines 1800-1810).

(3) The de-parsing secton (see Lines 2030-2500) is an offspring of Applesoft's LIST routine, modified to pring a single program line rather than an entire listing. I also eliminated the code which adds those extra blanks in the middle of quoted strings which take more than one screen line to LIST. To me it seems pretty neat!

Since I did not make any test to determine whether or not the program is RUNning at the time the slash is trapped in my filter, you have to be careful about using the slash character in REM statements. For example, "REM /150" will clear the screen and list line 150 at the top before proceeding. Other combinations of "/" in REM's may blow up. Also, typing "/" when Applesoft is executing an INPUT statement is now dangerous. Anyone know how to fix this?

```
        1000 *---------------------------------
        1010 *           LINE.EDIT
        1020 *
        1030 *        BY SANDY MOSSBERG
        1040 *
        1050 *   COMMERCIAL RIGHTS RESERVED
        1060 *
        1070 *---------------------------------
        1080 * 1.PACKS PROGRAM LINE FOR EASY EDITING.
        1090 *
        1100 * 2.USES CHRGET/CHRGOT FILTER ROUTINE NOTED IN AAL 9/81.
        1110 *
        1120 * 3.CHARACTER OUTPUT ROUTINE MODIFIED FROM APSOFT ROM
        1130 *   CODE (LIST, $D6A5-$D765).
        1140 *
        1150 * 4.INSTALLATION AND USE:
        1160 *    (A) BRUN LINE.EDIT.
        1170 *    (B) COMMAND "/LINENUMBER" PRODUCES PACKED LINE AT
        1180 *        TOP OF SCREEN.
        1190 *    (C) IF CHRGET/CHRGOT VECTOR DESTROYED BY APSOFT
        1200 *        COLDSTART (]FP, *E000G, *CTL-B), RESET LINE.EDIT
        1210 *        VECTOR BY CALL 768.
        1220 *---------------------------------
        1230            .OR $300
        1240 *---------------------------------
        1250 *          APPLESOFT POINTERS
        1260 *---------------------------------
0085-   1270 AS.FORPNT  .EQ  $85      ;HOLD Y-REGISTER
009B-   1280 AS.LOWTR   .EQ  $9B,$9C  ;LOCATION OF CHARACTER OR TOKEN IN PGM
009D-   1290 AS.DSCTMP  .EQ  $9D,$9E  ;LOCATION IN KEYWORD TABLE
        1300 *---------------------------------
        1310 *          APPLESOFT CHRGET/CHRGOT
        1320 *---------------------------------
00B1-   1330 AS.CHRGET  .EQ  $B1      ;GETS CHARACTER AT TEXT POINTER
00B8-   1340 AS.TXTPTR  .EQ  $B8,$B9  ;TEXT POINTER
00BA-   1350 AS.CHREXT  .EQ  $BA      ;CHRGET/CHRGOT VECTOR TO LINE.EDIT
00BE-   1360 AS.CHRENT  .EQ  $BE      ;RE-ENTRY TO CHRGET/CHRGOT
        1370 *---------------------------------
        1380 *          APPLESOFT ROM
        1390 *---------------------------------
D61A-   1400 AS.FNDLIN  .EQ  $D61A    ;ADDR NMBR IN LINNUM ($50,$51) TO LOWTR
DAFB-   1410 AS.CRDO    .EQ  $DAFB    ;LINEFEED
DB57-   1420 AS.OUTSP   .EQ  $DB57    ;OUTPUT SPACE
DB5C-   1430 AS.OUTDO   .EQ  $DB5C    ;OUTPUT CHARACTER
DD7B-   1440 AS.FRMEVL  .EQ  $DD7B    ;FORMULA AT TEXT POINTER TO FAC ($9D-$A2)
E752-   1450 AS.GETADR  .EQ  $E752    ;FAC TO INTEGER IN LINNUM ($50,$51)
ED24-   1460 AS.LINPRT  .EQ  $ED24    ;PRINT DECIMAL OF (A,X)
        1470 *---------------------------------
        1480 *          MONITOR ROM
        1490 *---------------------------------
FB5B-   1500 MON.TABV   .EQ  $FB5B    ;VTAB TO VALUE IN (A)
FC58-   1510 MON.HOME   .EQ  $FC58    ;HOME CURSOR, CLEAR SCREEN
FF3A-   1520 MON.BELL   .EQ  $FF3A    ;BEEP!
```

```
                 1540 *--------------------------------
                 1550 *  PUT LINE.EDIT VECTOR INTO CHRGET/CHRGOT
                 1560 *--------------------------------
0300- A9 4C      1570 START  LDA #$4C        ;JMP 'LINE.EDIT'
0302- 85 BA      1580        STA AS.CHREXT
0304- A9 0D      1590        LDA #EDIT
0306- 85 BB      1600        STA AS.CHREXT+1
0308- A9 03      1610        LDA /EDIT
030A- 85 BC      1620        STA AS.CHREXT+2
030C- 60         1630 RTS1   RTS
                 1640 *--------------------------------
                 1650 *  CHECK FOR VALID COMMAND
                 1660 *--------------------------------
030D- C9 2F      1670 EDIT   CMP #$2F        ;IS IT A SLASH (/)?
030F- D0 04      1680        BNE .1          ;NO. RETURN
0311- E6 B8      1690        INC AS.TXTPTR   ;YES. BUMP TEXT POINTER
0313- D0 07      1700        BNE .2          ;BRANCH ALWAYS
                 1710 *--------------------------------
                 1720 *  RETURN TO CHRGET/CHRGOT OR CALLER
                 1730 *--------------------------------
0315- C9 3A      1740 .1     CMP #$3A        ;IF COLON (EOS), SET Z AND C
0317- B0 F3      1750        BCS RTS1        ; FLAGS AND RETURN TO CALLER
0319- 4C BE 00   1760        JMP AS.CHRENT   ;IF NOT EOS, RE-ENTER CHRGET/CHRGOT
                 1770 *--------------------------------
                 1780 *  FIND LOCATION OF LINE NUMBER
                 1790 *--------------------------------
031C- 20 7B DD   1800 .2     JSR AS.FRMEVL   ;PUT LINE NUMBER INTO FAC ($9D-$A2)
031F- 20 52 E7   1810        JSR AS.GETADR   ;PUT FAC INTO LINNUM ($50,$51)
0322- 20 1A D6   1820        JSR AS.FNDLIN   ;PUT ADDR OF LINE INTO LOWTR
0325- 90 27      1830        BCC .5          ;CARRY CLEAR IF LINE NMBR NOT FOUND
                 1840 *--------------------------------
                 1850 *  CLEAR SCREEN AND SET TO ROW 2, COLUMN 2
                 1860 *--------------------------------
0327- 20 58 FC   1870        JSR MON.HOME
032A- 20 FB DA   1880        JSR AS.CRDO
032D- 20 57 DB   1890        JSR AS.OUTSP
                 1900 *--------------------------------
                 1910 *  PRINT LINE NUMBER
                 1920 *--------------------------------
0330- A0 02      1930        LDY #02         ;SET INDEX TO LINE NUMBER BYTES
0332- B1 9B      1940        LDA (AS.LOWTR),Y ;PUT LINE NUMBER LO
0334- AA         1950        TAX             ; INTO (X)
0335- C8         1960        INY
0336- B1 9B      1970        LDA (AS.LOWTR),Y ;PUT LINE NUMBER HI INTO (A)
0338- 84 85      1980        STY AS.FORPNT   ;HOLD (Y)
033A- 20 24 ED   1990        JSR AS.LINPRT   ;PRINT DECIMAL OF (A,X)
                 2000 *--------------------------------
                 2010 *  GET CHARACTER OR TOKEN
                 2020 *--------------------------------
033D- A9 20      2030        LDA #$20        ;SPACE
033F- A4 85      2040 .3     LDY AS.FORPNT   ;RESTORE (Y)
0341- 20 5C DB   2050 .4     JSR AS.OUTDO    ;PRINT CHARACTER IN (A)
0344- C8         2060        INY
0345- B1 9B      2070        LDA (AS.LOWTR),Y ;GET CHARACTER OR TOKEN
0347- D0 13      2080        BNE .8          ;IF NOT EOS (0), GET MORE
```

```
                  2100 *--------------------------------
                  2110 *  TWO ENDINGS — ONE HAPPY, ONE SAD
                  2120 *--------------------------------
0349- A9 00       2130        LDA #00          ;LINE WAS FOUND. END WITH
034B- 4C 5B FB 2140        JMP MON.TABV      ; CURSOR AT ROW 2, COLUMN 2
034E- 20 3A FF 2150 .5     JSR MON.BELL      ;LINE WAS NOT FOUND. END WITH
0351- 4C FB DA 2160        JMP AS.CRDO       ; CURSOR BELOW COMMAND INPUT
                  2170 *--------------------------------
                  2180 *  GET CHARACTER IN KEYWORD TABLE
                  2190 *--------------------------------
0354- C8          2200 .6     INY
0355- D0 02       2210        BNE .7
0357- E6 9E       2220        INC AS.DSCTMP+1
0359- B1 9D       2230 .7     LDA (AS.DSCTMP),Y
035B- 60          2240        RTS
                  2250 *--------------------------------
                  2260 *  PRINT CHARACTER OR KEYWORD
                  2270 *--------------------------------
035C- 10 E3       2280 .8     BPL .4            ;NON-TOKEN IS POS ASCII
035E- 38          2290        SEC               ;TOKEN MINUS $7F EQUALS INDEX TO
035F- E9 7F       2300        SBC #$7F          ; LOCATION OF KEYWORD IN TABLE
0361- AA          2310        TAX               ;PUT INDEX IN (X)
0362- 84 85       2320        STY AS.FORPNT     ;HOLD (Y)
0364- A0 D0       2330        LDY #$D0          ;KEYWORD TABLE STARTS AT $D0D0
0366- 84 9D       2340        STY AS.DSCTMP
0368- A0 CF       2350        LDY #$CF
036A- 84 9E       2360        STY AS.DSCTMP+1
036C- A0 FF       2370        LDY #$FF          ;WHEN BUMPED, (Y) WILL BE ZERO
036E- CA          2380 .9     DEX               ;DEC INDEX TO KEYWORD LOCATION
036F- F0 07       2390        BEQ .11           ;WHEN (X) IS ZERO, KEYWORD LOCATED
0371- 20 54 03 2400 .10    JSR .6            ;GET CHARACTER IN KEYWORD TABLE
0374- 10 FB       2410        BPL .10           ;IF POS ASCII, GET ANOTHER
0376- 30 F6       2420        BMI .9            ;IF NEG ASCII, DEC LOCATION INDEX
0378- 20 57 DB 2430 .11    JSR AS.OUTSP      ;PRINT SPACE
037B- 20 54 03 2440 .12    JSR .6            ;GET CHARACTER IN KEYWORD TABLE
037E- 30 05       2450        BMI .13           ;IT'S THE FINAL CHAR IN KEYWORD
0380- 20 5C DB 2460        JSR AS.OUTDO      ;PRINT NON-FINAL CHAR (POS ASCII)
0383- D0 F6       2470        BNE .12           ;BRANCH ALWAYS
0385- 20 5C DB 2480 .13    JSR AS.OUTDO      ;PRINT FINAL CHAR (NEG ASCII)
0388- A9 20       2490        LDA #$20          ;SPACE
038A- D0 B3       2500        BNE .3            ;BRANCH ALWAYS
                  2510 *--------------------------------
008C-             2520 SIZE   .EQ *-START
```

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOLKIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.
$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.
$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.
$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output froms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.
$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.
$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.
$15 Disk, Applesoft (32K, ROM or Language Card).

(Add $4.00 for Foreign Mail)

***Apple II is a registered trademark of the Apple Computer Co.**

Improved Applesoft Fast String Input....Bob Sander-Cederlof

In the April 1981 issue of AAL I printed a subroutine to read a line from the keyboard or a text file into an Applesoft string. The original version had a minor flaw (or major, if you happened to run into it): it left the high-order bit on in each byte, so that Applesoft could not compare them properly with strings from other sources. I printed a correction in a later issue, which stripped off the leading bit from each byte before putting it in the string.

Now Sherm Ostrowsky (from Goleta, California) has pointed out a more elegant solution. He uses a subroutine inside Applesoft that reads a line, terminates it with hex 00, and strips off the leading bit from each byte. The subroutine starts at $D52C. The only thing it doesn't do that we need is give us the length of the input line. Here is a commented listing of it.

```
                  1000 *-------------------------------
                  1010 *      APPLESOFT LINE INPUT SUBROUTINE
                  1020 *-------------------------------
                  1030        .OR $D52C
                  1040        .TA $82C
                  1050 *-------------------------------
0033-             1060 MON.PROMPT .EQ $33
FD6A-             1070 MON.RDLINE .EQ $FD6A
0200-             1080 BUFFER     .EQ $200
                  1090 *-------------------------------
                  1100 AS.INLINE
D52C- A2 80       1110        LDX #$80     NULL CHARACTER
D52E- 86 33       1120 INLIN2 STX MON.PROMPT  FOR THE PROMPT CHARACTER
D530- 20 6A FD    1130        JSR MON.RDLINE  READ A LINE INTO BUFFER
D533- E0 EF       1140        CPX #239     TRUNCATE TO 239 CHARACTERS
D535- 90 02       1150        BCC .1
D537- A2 EF       1160        LDX #239
D539- A9 00       1170 .1     LDA #0       MARK END OF LINE WITH $00
D53B- 9D 00 02    1180        STA BUFFER,X
D53E- 8A          1190        TXA          # REAL CHARS IN LINE
D53F- F0 0B       1200        BEQ .3       EMPTY LINE
D541- BD FF 01    1210 .2     LDA BUFFER-1,X  STRIP OFF ALL SIGN BITS
D544- 29 7F       1220        AND #$7F
D546- 9D FF 01    1230        STA BUFFER-1,X
D549- CA          1240        DEX
D54A- D0 F5       1250        BNE .2
D54C- A9 00       1260 .3     LDA #0
D54E- A2 FF       1270        LDX #BUFFER-1
D550- A0 01       1280        LDY /BUFFER-1
D552- 60          1290        RTS
```

Since $D52C stores $80 (null) in the prompt character, you might want to load the X-register with $87 (bell) and enter at $D52E instead.

Since the subroutine returns with $FF in the X-register, and we need the length of the input line instead, we can use the following code to get the line length in X:

```
        JSR $D52C
.1      INX
        LDA $200,X
        BNE .1
```

Here is a new version, then, of my fast string input subroutine:

```
                  1000 *-------------------------------
                  1010 *      FAST STRING INPUT ROUTINE
                  1020 *      &GET <STRING VARIABLE>
                  1030 *      ACCEPTS ANY CHARACTER, UNLIKE NORMAL INPUT
                  1040 *-------------------------------
                  1050        .OR $300
                  1060        .TF B.FAST READ
                  1070 *-------------------------------
00B1-             1080 AS.CHRGET  .EQ $00B1
DEC9-             1090 AS.SYNERR  .EQ $DEC9
D52C-             1100 AS.INLINE  .EQ $D52C
DFE3-             1110 AS.PTRGET  .EQ $DFE3
E452-             1120 AS.GETSPA  .EQ $E452
E5E2-             1130 AS.MOVSTR  .EQ $E5E2
                  1140 *-------------------------------
0071-             1150 ADDR       .EQ $71 AND 72
0083-             1160 PNTR       .EQ $83 AND 84
009D-             1170 LENGTH     .EQ $9D
0200-             1180 BUFFER     .EQ $200
                  1190 *-------------------------------
0300- C9 BE       1200 GET    CMP #$BE     "GET" TOKEN
0302- F0 03       1210        BEQ .1       YES
0304- 4C C9 DE    1220        JMP AS.SYNERR  SORRY...
0307- 20 B1 00    1230 .1     JSR AS.CHRGET SET UP THE FOLLOWING CHARACTER
030A- 20 E3 DF    1240        JSR AS.PTRGET FIND THE STRING VARIABLE POINTER
030D- 20 2C D5    1250        JSR AS.INLINE READ A LINE INTO BUFFER
0310- E8          1260 .2     INX          COMPUTE THE LENGTH OF THE LINE
0311- BD 00 02    1270        LDA BUFFER,X
0314- D0 FA       1280        BNE .2       NOT AT END OF LINE YET
0316- 86 9D       1290        STX LENGTH   SAVE LINE LENGTH
0318- 8A          1300        TXA
0319- 20 52 E4    1310        JSR AS.GETSPA  GET SPACE IN STRING AREA
031C- A0 00       1320        LDY #0       SET UP STRING VARIABLE POINTER
031E- 91 83       1330        STA (PNTR),Y   LENGTH
0320- C8          1340        INY
0321- A5 71       1350        LDA ADDR
0323- 91 83       1360        STA (PNTR),Y   ADDRESS (LO-BYTE)
0325- C8          1370        INY
0326- A5 72       1380        LDA ADDR+1
0328- 91 83       1390        STA (PNTR),Y   ADDRESS (HI-BYTE)
032A- A0 02       1400        LDY /BUFFER    SET UP TO COPY STRING DATA
032C- A2 00       1410        LDX #BUFFER    INTO STRING AREA
032E- A5 9D       1420        LDA LENGTH
0330- 4C E2 E5    1430        JMP AS.MOVSTR  COPY IT NOW, AND RETURN
```

Here is how you might use it from an Applesoft program, to read a series of lines from a file:

```
     100 D$ = CHR$ (4)
     110 PRINT D$"BLOAD B.FAST READ"
     120 POKE 1013,76 : POKE 1014,0 : POKE 1015,3
     210 PRINT D$"OPEN MY.FILE"
     220 PRINT D$"READ MY.FILE"
     230 FOR I = 1 TO 10
     240 & GET A$(I)
     250 NEXT I
```

Note that the subroutine is fully relocatable. Since there are no internal JMP's or JSR's, and no internal variables, you can load the program anywhere it will fit and run it without any modifications. Just be sure to change line 120 above to POKE the correct address in 1014 and 1015.

Note that this patch does not "print" the ASCII codes on the screen; it "pokes" them. Therefore if your printer is on, the printed copy will only contain the hex dump. The ASCII codes will only appear on the screen.

How do you patch the RAM card version of the monitor? Here's how I did it:

1) Load the language card using your DOS 3.3 Master Disk, or whatever technique you like to use.

2) Turn on the language that is in the card (using FP or INT).

3) BSAVE MONITOR,A$F800,L$800.

4) BRUN ASMDISK 4.0

5) BLOAD MONITOR,A$800

6) Enter the source code for the patches and assemble them with the ASM command. This will patch the monitor copy which you loaded at A$800 in step 5.

Adding ASCII to Apple Monitor Dump....Bob Sander-Cederlof

Peter Bartlett (subscriber in Chicago, IL) sent me some source code for patches to the Apple Monitor ROM. Of course, patching a ROM may be a little too much hardware work, but if you have a 16K RAM card you can put the revised monitor up there. The space needed for the patch is stolen from the cassette I/O command, so if you install this patch you will lose cassette I/O.

Peter's patches add the ASCII dump to the Apple Monitor's hex dump. That is, when I type a command like "800.87F" in the monitor, it will not only print out the hex values, but also the ASCII values of each byte. I modified his patches a little, to shorten the code to the following:

```
                1000 *-------------------------------
                1010 *         PATCHES TO ADD ASCII DUMP
                1020 *         TO THE APPLE MONITOR
                1030 *-------------------------------
003C-           1040 A1L           .EQ $3C
FDED-           1050 COUT          .EQ $FDED
                1060 *-------------------------------
                1070           .OR $FDB8
                1080           .TA $0DB8
FDB8- 20 C9 FC  1090           JSR PATCH    CALL MY PATCH CODE
                1100 *-------------------------------
                1110           .OR $FCC9
                1120           .TA $0CC9
                1130 PATCH
FCC9- 20 ED FD  1140           JSR COUT     PRINT A SPACE
FCCC- B1 3C     1150           LDA (A1L),Y  GET BYTE TO BE DISPLAYED
FCCE- 48        1160           PHA          SAVE IT ON STACK
FCCF- A5 3C     1170           LDA A1L      LOW BYTE OF DUMP ADDRESS
FCD1- 29 07     1180           AND #7       MASK LINE POSITION
FCD3- 18        1190           CLC
FCD4- 69 1F     1200           ADC #31      COMPUTE HORIZONTAL OFFSET
FCD6- A8        1210           TAY
FCD7- 68        1220           PLA          GET BYTE FROM STACK
FCD8- 91 28     1230           STA ($28),Y  STORE IT ON THE SCREEN
FCDA- A0 00     1240           LDY #0       RESTORE Y
FCDC- 60        1250           RTS
```

These patches will work with either the old monitor ROM, or the Autostart ROM. The JSR PATCH line goes right into the hex dump program, over the top of a JSR COUT that printed a space. That space is normally printed right before the next byte value is printed in hex. The address of the next byte is kept in A1L,A1H ($3C,3D). The Y-register has 0 in it.

The main patch subroutine is stored on top of part of the cassette tape I/O, at $FC99; it begins with the JSR COUT that was covered up at $FDB8. Lines 1150,1160 pick up the byte to be displayed and save it on the stack. Lines 1170-1210 compute the horizontal postition for poking the byte on the screen. The low-order three bits of the memory address determine which column will be used, from column 31 through 38. Lines 1220,1230 retrieve the byte from the stack and store it into the screen buffer. Lines 1240,1250 restore Y=0 and return to the hex dump subroutine.

7) Type "$C081 C081" to write enable the language card.

8) Type "$F800<800.FFFM" to move the patched monitor into the real monitor space.

9) Type "BSAVE <your file name>,A$D000,L$3000" to save the combined language and monitor for later loading into the language card.

If you really do want to burn a new monitor ROM, follow the instructions with your ROM Burner.

Applesoft GOTO from Assembly Language.........Bob Sander-Cederlof

Bob Potts called the other day with an interesting question. Suppose you want to jump to a particular line (by line number) of an Applesoft program, rather than simply returning from an assembly language program.

For example, I might call an assembly language subroutine at $300 with "CALL 768". After it does its job, the subroutine may decide either to return to the following Applesoft statement by an "RTS" instruction, or to GOTO a particular line number in the program. (Perhaps an error processing subroutine in the Applesoft code.) Can it be done?

Yes, and it is fairly simple. First we need to put the binary value of the line number into locations $50 and $51. Then we must jump to $D944 in the Applesoft ROMs to finish the GOTO operation. Here is the code to jump to line number 1350, for example:

```
GOTO1350 LDA #1350   LOW BYTE OF "1350"
         STA $50
         LDA /1350   HIGH BYTE OF "1350"
         STA $51
         JMP $D955   APPLESOFT GOTO PROCESSOR
```

That's all there is to it!

I wrote a tiny little subroutine to demonstrate that this works. It expects to find the line number in $2FE and $2FF. You can POKE it there before CALLing 768. Here is my subroutine:

```
                  1000 *--------------------------------
                  1010 *       GO TO <LINE #>
                  1020 *       POKE THE LINE # INTO 766,767
                  1030 *       AND CALL768 TO GO TO IT
                  1040 *--------------------------------
                  1050         .OR $300
0300- AD FE 02 1060 GOTO    LDA $2FE
0303- 85 50    1070         STA $50
0305- AD FF 02 1080         LDA $2FF
0308- 85 51    1090         STA $51
030A- 4C 44 D9 1100         JMP $D944
```

Now here is a test program in Applesoft. Can you tell what it will do before you try it? The first two lines poke in the GOTO subroutine. The next five lines call the subroutine for successive values 1000, 2000, 3000 etc. up to 9000. The code in line 10000 jumps back to line 140 to continue the loop. Try it!

```
10  FOR I = 0 TO 12: READ A: POKE 768 + I,A: NEXT
20  DATA 173,254,2,133,80,173,255,2,133,81,76,68,217
100  FOR I = 1000 TO 9000 STEP 1000
110 IH =  INT (I / 256):IL = I - IH * 256
120  POKE 766,IL: POKE 767,IH
130  CALL 768
140  NEXT I
150  END
1000  PRINT 1000: GOTO 10000
2000  PRINT 2000: GOTO 10000
3000  PRINT 3000: GOTO 10000
4000  PRINT 4000: GOTO 10000
5000  PRINT 5000: GOTO 10000
6000  PRINT 6000: GOTO 10000
7000  PRINT 7000: GOTO 10000
8000  PRINT 8000: GOTO 10000
9000  PRINT 9000
10000  POKE 766,140: POKE 767,0: CALL 768
```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage). All material herein is copyrighted by S-C SOFTWARE, all rights reserved. Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)